VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Bakalářská práce

2011 Jakub Lattenberk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra Informatiky

Detekce návrhových vzorů v kódech .NET

Design pattern detection in .NET programming code

2011 Jakub Lattenberk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Lattenberk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Detekce návrhových vzorů ve zdrojových kódech .NET
Design Pattern Detection in .NET Programming Code

Zásady pro vypracování:

Cílem práce je počítačová implementace vybraného algoritmu pro detekci návrhových vzorů ve zdrojových kódech a provedení experimentů s testovacími i reálnými daty.

Dílčí cíle:
1. Seznámit se s existujícími přístupy a tyto přístupy v práci popsat.
2. Vybrat jeden z existujících přístupů.(algoritmů).
3. Implementovat aplikaci realizující vybraný algoritmus, aplikaci v práci popsat.
4. Provést a popsat experimenty s testovacími a reálnými daty.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 19.11.2011

Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum Podpis

Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

Datum Podpis

Abstrakt

Cílem této práce je nahlédnout do problematiky detekce návrhových vzorů v kódech .NET. Seznámení se s problematikou převodu návrhového vzoru do grafu a následnou reprezentaci grafu jako matice generalizace a asociace. Pro detekci návrhových vzorů je v práci použit a detailně popsán algoritmus Similarity Scoring.

Klíčová slova

Návrhový vzor, detekce, .NET, reflexe, CIL, graf, matice, podobnost, podobnostní bodování, generalizace, asociace

Abstract

The aim of this thesis is to look into the issue of design pattern detection in .NET Programming code. Introduction into the issue of treansfer of design patterns to a graph and representation of graph as matrices of generalization and association. For detection of design patterns is in the thesis used and described in detail the Similarity Scoring algorithm.

Key words

Design patterrn, detection, .NET, reflection, CIL ,graph, matrices, similarity, similarity scoring, generalization, association

Obsah

1. Úvod

Návrhové vzory představují obecné řešení problému. Typicky ukazují vztahy a interakce mezi třídami a objekty. Jejich znalost a použití pomáhá při návrhu aplikací. Návrhové vzory popisují postupy, které se během programování často opakují a které se často používají k řešení stejného problému.

Identifikace návrhového vzoru, může být užitečná k pochopení návrhu a poskytuje základ pro další zlepšení. Může být také měřítkem kvality analyzovaného softwaru, jestli je založen na daných standardech.

První část je věnovaná metodám detekce a detailnímu popisu metody Similarity scoring.

Druhá část je věnována popisu aplikace.

2. Motivace

V dnešní době je vhodné posoudit a analyzovat vytvořené aplikace, jestli vývojáři při vývoji systému použili návrhové vzory, popisující postupy, které se během programování často opakují a často se používají k řešení stejného problému. Proto je cílem této bakalářské práce seznámit se s metodami detekce na platformě .NET. Práce se zaměřuje na dva druhy metod: přesné a nepřesné. V první kategorii se budeme zabývat izomorfizmem grafu a ve druhé si detailně popíšeme algoritmus Similarity Scoring (podobnostní bodování).

3. Návrhové vzory

Vznik návrhových vzorů se datuje do šedesátých let, kdy pro vzory v architektuře použil tento termín Christopher Alexander s kolegy. V počítačových vědách se návrhové vzory poprvé objevily na konferenci OOPSLA (Objec-Oriented Programming, Systems, Languages and Applications) v roce 1987 v Orlandu viz. (1)

3.1. Rozdělení návrhových vzorů

Návrhové vzory se dělí do tří skupin:

1. Vytvářející
2. Strukturální
3. Chování

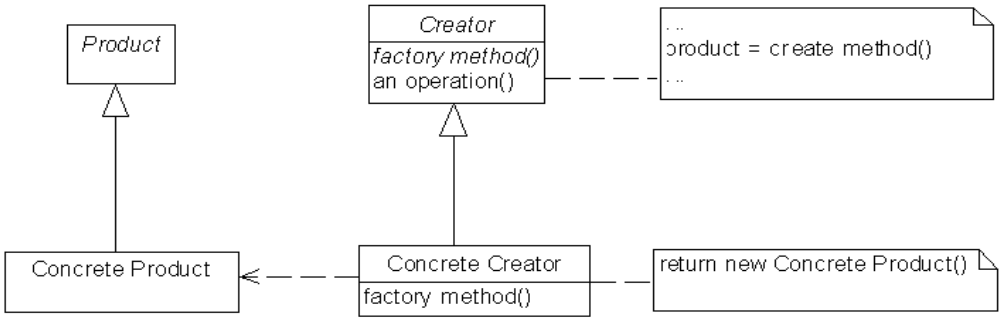
Dále jsou uvedeny ukázky návrhových vzorů, po jednom z každé kategorie.

3.1.1. Vytvářející

Řeší problémy s vytvářením objektů v systému. Snahou těchto vzorů je popsat postup při výběru třídy nového objektu a zajištění počtu těchto objektů. Mezi tyto vzory patří: Factory, Abstract Factory, Builder, Prototype.

Návrhový vzor Factory – Továrna

Předpokladem je existence několika tříd, které mají stejného předka, ale poskytují jiné služby nad jinými daty. Factory potom dovoluje vybrat v průběhu programu vytvoření instance některé z těchto tříd.



Obrázek 1 : Návrhový vzor factory

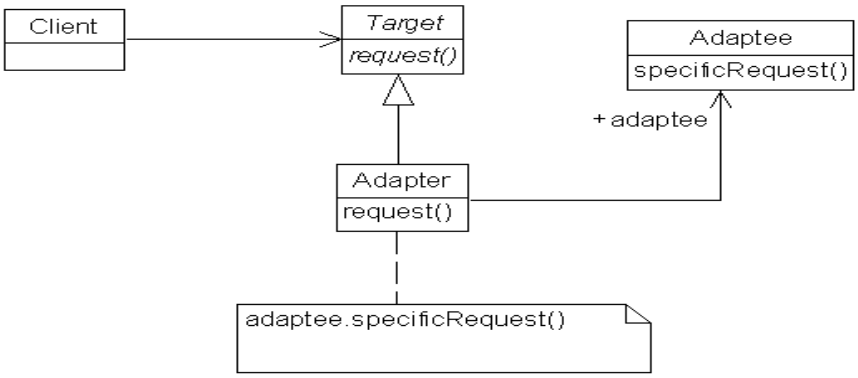
3.1.2. Strukturální

Představují skupinu návrhových vzorů, které se snaží zaměřit na strukturu systému, uspořádání tříd a komponent. Mezi tyto vzory patří: Adapter, Bridge, Composite, Decorator a další.

Návrhový vzor Adapter

Jedná se o přizpůsobení třídy, aby ji bylo možné používat i jiným požadovaným způsobem. Problémem je konverze rozhraní jedné třídy na rozhraní druhé.

3.1.3. Chování

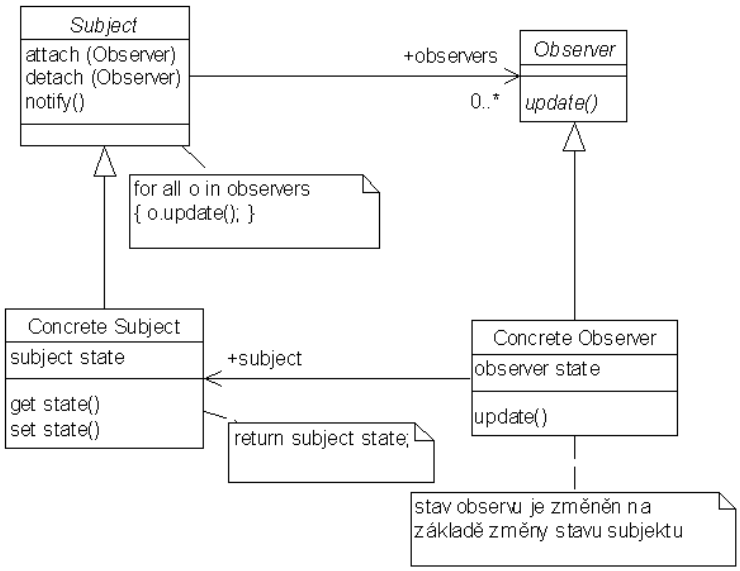


Obrázek 2 : Návrhový vzor adapter

Zajímají se o chování systému. Mohou být založeny na třídách nebo objektech. Mezi tyto vzory patří: Observer, Memento, Iterator a další.

Návrhový vzor Observer – pozorovatel

Observer je možné použít v situaci, kdy je definována závislost jednoho objektu na druhém. Závislost, ve smyslu tohoto návrhového vzoru, představuje propagaci změny nezávislého objektu závislým objektům (pozorovatelům).



Obrázek 3 : Návrhový vzor Observer

Každý Návrhový vzor je možné reprezentovat jako množinu vrcholů a hran grafu G.

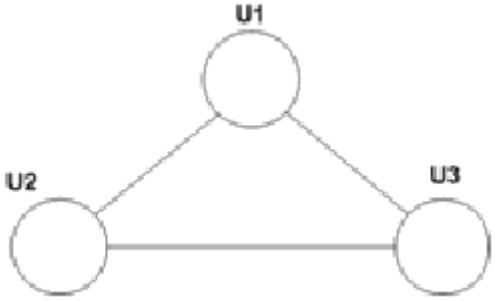
Více o návrhových vzorech lze najít v (2)

4. Teorie grafu

Grafy jsou vhodným prostředkem pro popis situací, které lze znázornit pomocí konečného množství bodů a vztahů mezi nimi znázorněných pomocí hran.

4.1. Definice 1 – Neorientovaný graf

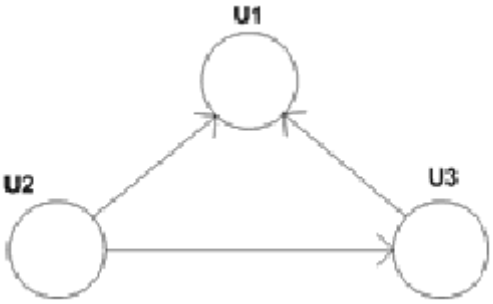
Graf (obyčejný, neorientovaný) je uspořádaná dvojice G=(V, E), kde V je množina vrcholů a E je množina hran – množina vybraných dvouprvkových podmnožin vrcholů.



Obrázek 4 : Neorientovaný graf

4.2. Definice 2 – Orientovaný graf

Orientovaný graf je uspořádaná dvojice G(V,E), kde V je množina vrcholů a E je podmnožina kartézského součinu množiny vrcholů.



Obrázek 5: Orientovaný graf

4.3. Isomorfizmus grafu

Grafy G a G‘ jsou isomorfní právě tehdy, když existuje takové zobrazení $f: V(G) \rightarrow V(G')$, že platí $\{i,j\} \in E(G) \Leftrightarrow \{f(i), f(j)\} \in E(G')$. Tedy zhruba řečeno: G a G‘ se liší pouze očíslováním vrcholů.

4.4. Reprezentace grafu jako matice

Grafy můžeme reprezentovat několika způsoby. Ten častější, zde již zobrazený je grafický, pomocí diagramu. Druhý způsob je pomocí matice.

Definice Matice sousednosti:

Nechť je G orientovaný graf na vrcholech $\{v_1, \dots, v_n\}$. Matice sousednosti grafu G je čtvercová matice o řádu n definovaná předpisem $S(G) = (\sigma_{i,j})$, kde $\sigma_{i,j} \begin{cases} 1\ pokud\ v_i v_j \in E(G) \\ 0\ jinak \end{cases}$ pro $i, j = 1, \dots, n$.

5. Převod vzoru do grafu

Na návrhový vzor můžeme pohlížet jako na orientovaný graf, kde jednotlivé uzly jsou třídy vzoru a hrany jsou vazby mezi nimi. Při převádění musíme tvořit grafy zvlášť pro generalizaci a zvlášť pro asociaci, protože operace mezi grafy jsou pak jednodušší. Generalizace je hierarchický vztah tříd, v němž třída – potomek dědí atributy a operace třídy – předka. Asociace je obecný sémantický vztah mezi prvky modelu, který specifikuje spojení mezi instancemi viz. (3). Dále pak orientované grafy převedeme na čtvercové matice, kdy počet řádků a sloupců matice je roven uzlů grafu (tříd návrhového vzoru).

Jako příklad si uvedeme převod návrhového vzoru Observer. Podle obrázku Obrázek 3**Chyba! Nenalezen zdroj odkazů.**. Návrhový vzor obsahuje 4 třídy a je v něm zastoupena jak dědičnost (generalizace), tak asociace. Takže nám vytvoří 2 orientované grafy, ze kterých sestavíme matice sousednosti.

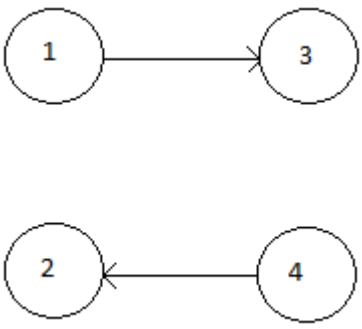
Nejprve si třídy vzoru číselně označíme, indexy značí číslo řádku a sloupce v matici:

Subject = 1

Concretesubject = 2

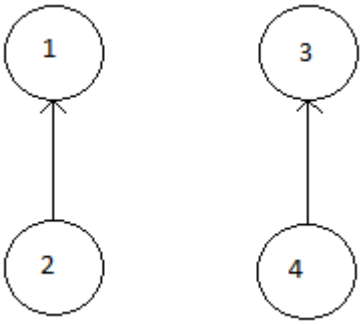
Observer = 3

ConcreteObserver = 4



Obrázek 6 : Graf vzoru Observer - asociace

$$A_{asociace} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



Obrázek 7 : Graf vzoru Observer - generalizace

$$A_{generalizace} = \begin{pmatrix} 0\,0\,0\,0 \\ 1\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix}$$

Převod grafu na matice sousednosti nám umožní další operace spojené s detekcí.

6. Metody detekce

Jakmile máme grafy popisující testovaný program a grafy popisující vzor, můžeme je začít porovnávat. V současné době existuje několik technik zabývajících se problematikou porovnávání grafů.

Hlavní rozdělení, které se uvádí v literatuře (4) ,je rozdělení na přesné a nepřesné metody, které se dále dělí do dalších podkategorií.

6.1. Přesné metody

Tyto metody jsou založeny na izomorfizmu grafu. Je nutné, aby oba grafy měly stejný počet uzlů. Většinou je systém mnohem větší než návrhový vzor. Z toho důvodu se musí v grafu systému hledat podgrafy o velikosti návrhového vzoru. Pak je možné grafy porovnávat a zjistit jestli jsou izomorfní.

Celá tato kategorie podobností grafů spadá do NP-úplných problémů, proto se některé nedají v konečném čase vyřešit.

Vstupem algoritmu jsou matice vzoru ($Patt$) a matice reprezentující systém (Sys)

Popis algoritmu

1. Vytvoření všech matic, z matice reprezentující analyzovaný systém, se záměnou sloupců a řádků.

2. Vytvoření všech pod-matic, z matice systému, o řádu matice reprezentující vzor.
3. Porovnání všech pod-matic systému s maticí vzoru.

6.2. Nepřesné metody

Tyto metody řeší problém složitosti u přesných metod. Nevýhodou je to, že nedokáží přesně říct, je-li vzor v kódu analyzovaného systému obsažen. Výhodou je že matice vzoru a systému nemusí mít stejný počet uzlů.

Pro implementaci aplikace, která je cílem této práce jsem si vybral metodu „Similarity scoring“ (metoda podobnostního bodování).

6.2.1. Editační vzdálenost

Editační vzdálenost mezi grafy, je definována jako počet modifikací grafu, pomocí kterých, lze jeden graf převést na druhý (přidáním nebo odebráním hran a uzlů). Jelikož graf systému je většinou rozsáhlejší než graf vzoru, bude převažovat odebírání hran a uzlů. Tím pádem se zjednoduší i složitost operace. Více o této metodě najdeme v (4).

6.2.2. Similarity scoring

Tento algoritmus jsem si vybral k implementaci aplikace. Je založen na výpočtu podobnostního skóre dvou matic, které udává, jak moc je v systému vzor zastoupen. Vychází ze vzorce, který počítá podobnostní matici tak dlouho, dokud se předchozí a právě vypočítaná matice neliší o předem definovanou hodnotu.

Vstupem výpočtu jsou matice A a B, popisující návrhový vzor a systém. Podobnostní matice S je definována jako $n_B \times n_A$, jejíž členy $S_{i,j}$ reprezentují jak moc je vrchol j grafu G_A, tedy podobnostní matice A, podobný vrcholu i grafu G_B tedy podobnostní matice B. Toto číslo se nazývá podobnostní skóre mezi dvěma vrcholy.

Popis algoritmu

1. Nastavíme hodnoty matice Z_0 na hodnotu 1
2. Opakujeme výpočet podle vzorce, dokud matice nekonvergují

$$Z_{k+1} = \frac{BZ_kA^T + B^TZ_kA}{\|BZ_kA^T + B^TZ_kA\|_1}$$

3. Výstupní matice S je poslední hodnotou matice Z_k

Vysvětlivky

- A, B jsou matice sousednosti grafů G_A, G_B
- Z_0 je $n_B \times n_A$ matice naplněná jedničkami
- $\|.\|_1$ je první norma matice

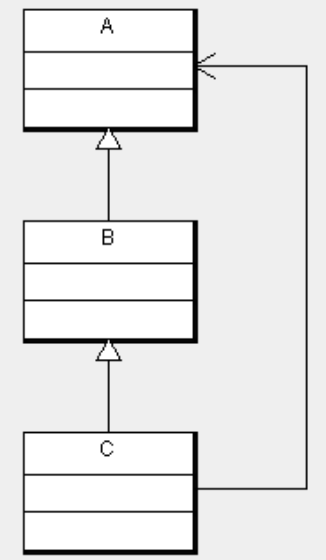
Výpočet první normy

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{i,j}|$$

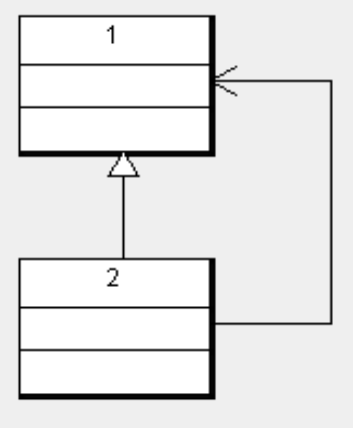
První norma vypočítá maximální hodnoty sloupců v matici. Viz. (5)

Příklad:

Na jednoduchém příkladu si ukážeme, jak tato metoda funguje:



Obrázek 9 : Část systému



Obrázek 8 : Návrhový vzor

Systém a vzor na matice sousednosti

$$Sys_{generalizace} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad Patt_{generalizace} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

$$Sys_{asociace} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad Patt_{asociace} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Nejprve provedeme výpočet s maticemi asociace:

Vstup

- Matice asociace reprezentující návrhový vzor
- Matice asociace reprezentující systém

Výstup

$$S_{asociace} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Dále provedeme výpočet s maticemi generalizace:

Vstup

- Matice generalizace reprezentující návrhový vzor
- Matice asociace reprezentující systém

$$S_{generalizace} = \begin{pmatrix} 0.5 & 0 \\ 0.5 & 0.5 \\ 0 & 0.5 \end{pmatrix}$$

Nyní máme k dispozici dvě matice, matice reprezentující podobnostní skóre návrhového vzoru a analyzovaného systému, za použití asociace a generalizace. Výslednou matici dostaneme tak, že matice sečteme a součet normalizujeme. Normalizovaná matice je taková, které má prvky na hlavní diagonále rovny $1/k$, kde k je počet matic, ve kterých byl popsán návrhový vzor. V našem případě $k = 2$.

Výsledná matice tedy bude:

$$S_{generalizace} = \begin{pmatrix} 0.75 & 0 \\ 0.25 & 0.25 \\ 0 & 0.75 \end{pmatrix}$$

Indexy sloupců podobnostní matice S reprezentují třídy návrhového vzoru a indexy řádků třídy analyzovaného systému. Budeme-li matici analyzovat, zjistíme, že z matice vyplývá silná podobnost mezi třídami A-1 a C-2. Podle obrázků Obrázek 8 a Obrázek 9, můžeme říct, že systém a návrhový vzor jsou vzájemně podobné.

7. .NET

Pro analýzu můžeme použít jakýkoli objektově orientovaný jazyk, jako je například C# nebo C++, ale často se stává, že komponenty systému jsou napsány v různých jazycích, což je pro analýzu nevhodné. Z toho důvodu je za potřebí, použít nějaká framework, který tyto různé vyšší jazyky přeloží do nízko-úrovňového jazyka, který je pro ty vyšší jazyky stejný.

Tím je .NET Framework – jeden z nejrozšířenějších frameworků, jak už pro tvorbu desktopových aplikací, tak pro tvorbu webových..NET Freamework je součástí platformy .NET od společnosti Microsoft. Je plně objektový a obsahuje několik programovacích jazyků, jako je C#, C++, Visual Basic, J# nebo méně obvyklý F#. Zdrojový kód napsaný v jakémkoli uvedeném jazyce je přeložen do nízko-úrovňového jazyka Common Intermediate Language (CIL). 1. verze frameworku byla uvedena v roce 2002, nyní se nachází ve verzi 4 s vývojovým prostředím Visual Studio 2010.

7.1. Common Intermediate Language

Při kompilování .NET programovacích jazyků je zdrojový kód přeložen do CIL kódu. CIL je procesorově a platformě nezávislý soubor instrukcí, které mohou být realizovány v jakémkoli prostředí s nainstalovaným .NET Frameworkem. CIL je za běhu ošetřován z hlediska bezpečnosti, a proto poskytuje lepší zabezpečení a spolehlivost než nativně kompilované binární soubory viz. (6). Mezi jeho základní vlastnosti patří:

- Objektová orientace
- Přísná typizace dat
- Ošetření chyb prostřednictvím výjimek
- Užití atributů

7.2. Reflexe

Mnohé ze služeb .NET závisí na přítomnosti metadat. Vytvořené programy mohou využívat tyto metadata a rozšiřovat je o nové informace. Reflexí je označováno prozkoumání existujících typů prostřednictvím metadat. Uskutečňuje se tak prostřednictvím sady typů v oboru názvů *System.Reflection*. Reflexe představuje procházení a manipulování s objektovým modelem, který představuje nějakou aplikaci.

Mezi hlavní a pro účel vytvoření grafů asociace a generalizace patří dvě metody. První z nich je metoda *GetTypes()*. Metoda vrátí kolekci typů, definovaných v dané assembly, díky kterým budeme schopni sestavit grafy reprezentující asociaci a generalizaci. Druhou metodou je metoda *GetFields()*. Metoda nám vrátí kolekci datových typů, které assembly obsahuje. Tyto data opět využijeme při sestavování grafů.

8. Uživatelská příručka

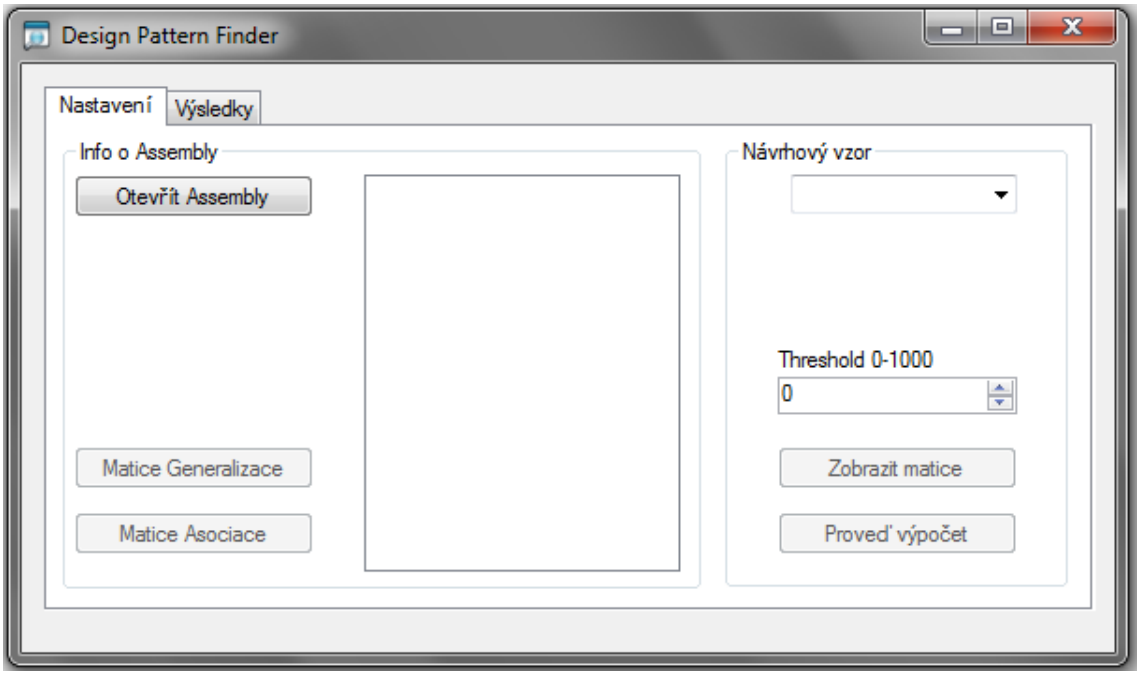
Pro detekci návrhových vzorů v kódech .NET jsem vytvořil aplikaci Design Pattern Finder, které implementuje metodu Similarity Scoring. Jedná se o jednoduchou desktopovou aplikaci, která obsahuje několik návrhových vzorů vhodných pro detekci:

- Abstract factory
- Adapter
- Bridge
- Decorator
- Composite
- Observer

Dále nabízí zobrazení matic podobnosti načtených komponent nebo vybraného návrhového vzoru. Po dokončení výpočtu, jsou výsledky jednoduše zobrazeny v kartě „Výsledky“.

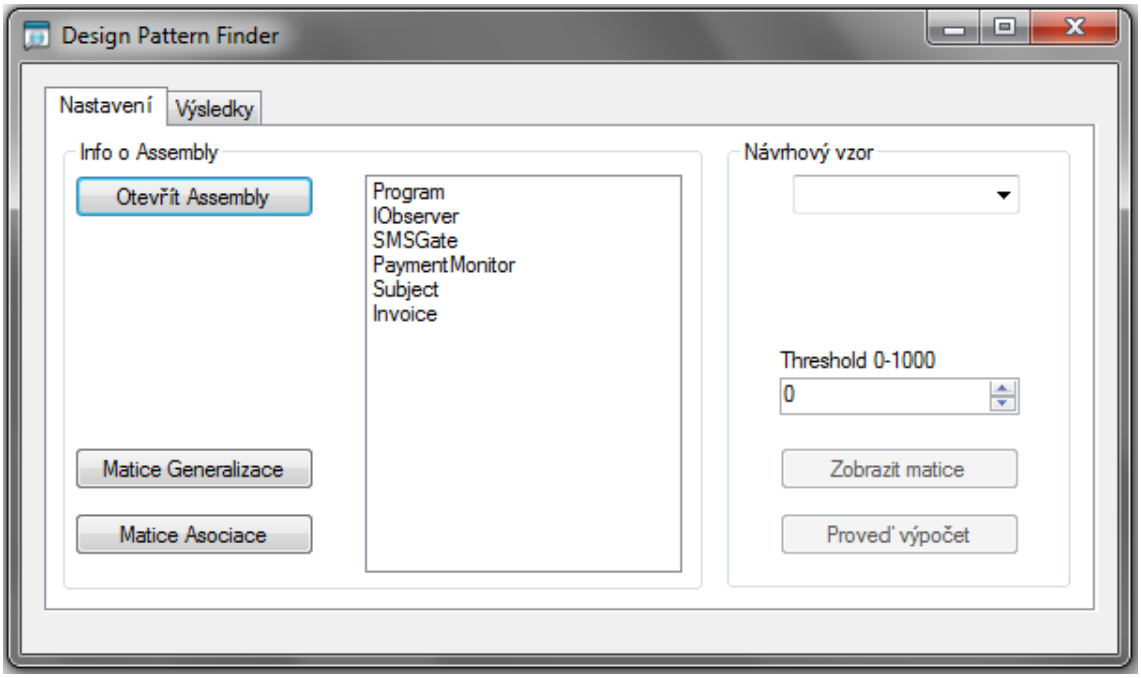
Obsahem uživatelské příručky je seznámit čtenáře s vlastním ovládáním aplikace a jejími vlastnostmi. Aplikace Design Pattern Finder je vytvořena jako klasická desktopová aplikace pro operační systém Microsoft Windows XP (SP3) a vyšší. Pro její spuštění je potřeba mít nainstalovaný .NET Framework 4 Client Profile. Aplikace nevyžaduje instalaci, proto je možné ji spouštět přímo.

Po spuštění se zobrazí hlavní okno (viz. Obrázek 11), které je rozděleno na dvě záložky, jednou pro nastavení analyzovaného systému a druhou pro zobrazení výsledků výpočtu.



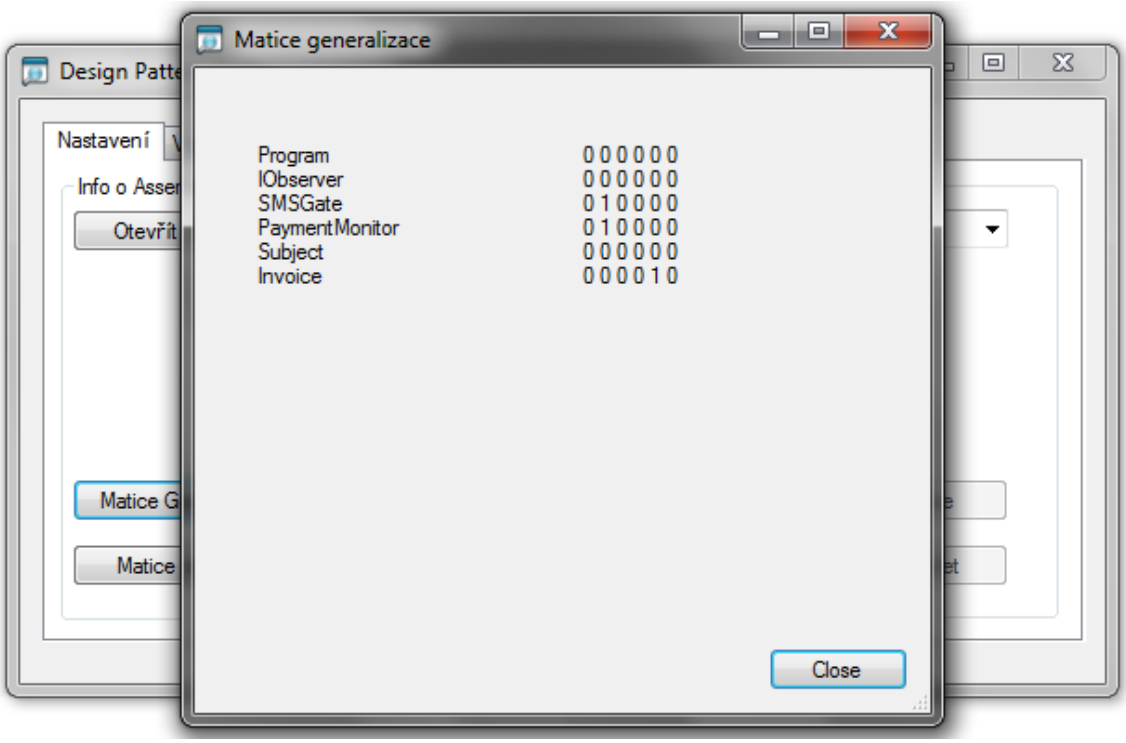
Obrázek 11 : Hlavní okno aplikace

Po načtení zvolené komponenty se zobrazí seznam tříd, v ní nalezených a pomocí tlačítek „Matice Generalizace“ a „Matice Asociace“ si můžeme zobrazit příslušné matice vazeb mezi třídami. (viz. Obrázek 10)

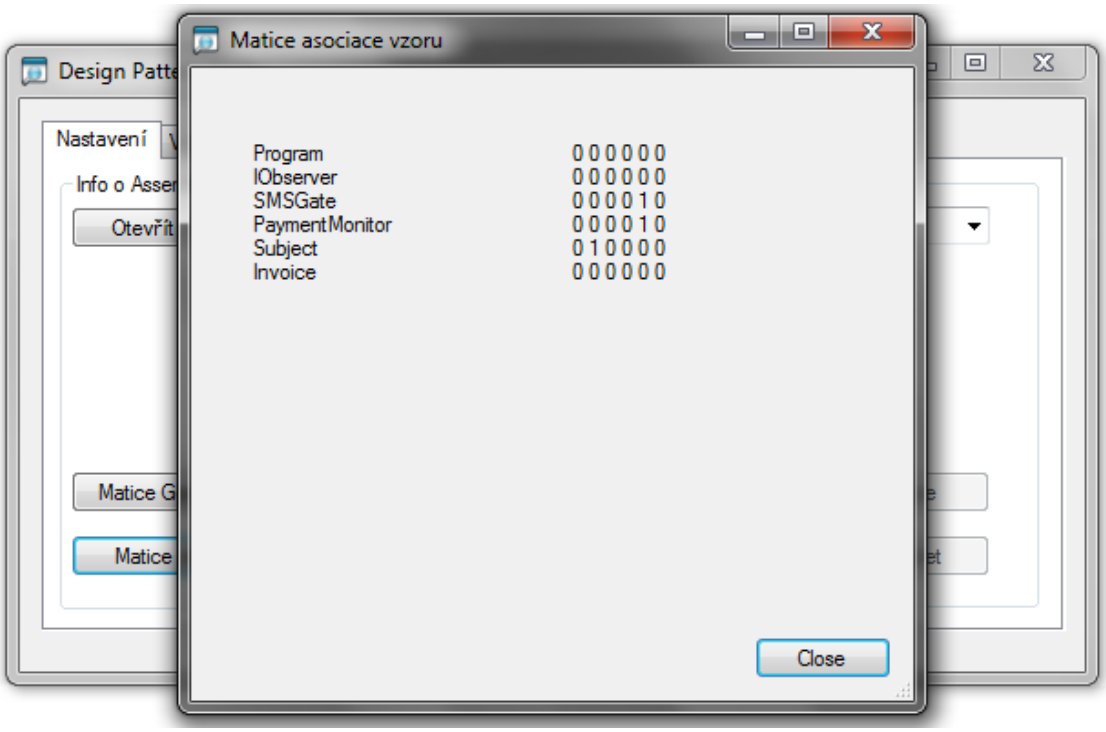


Obrázek 10 : Zobrazení tříd načtené komponenty

Po kliknutí na jedno z tlačítek se zobrazí okno s vybranou maticí. Matice je čtvercová a je u ní zobrazen popis tříd. V místě, kde je mezi třídami vazba (generalizace, asociace) je zobrazena jednička. (viz. Obrázek 12, Obrázek 13)

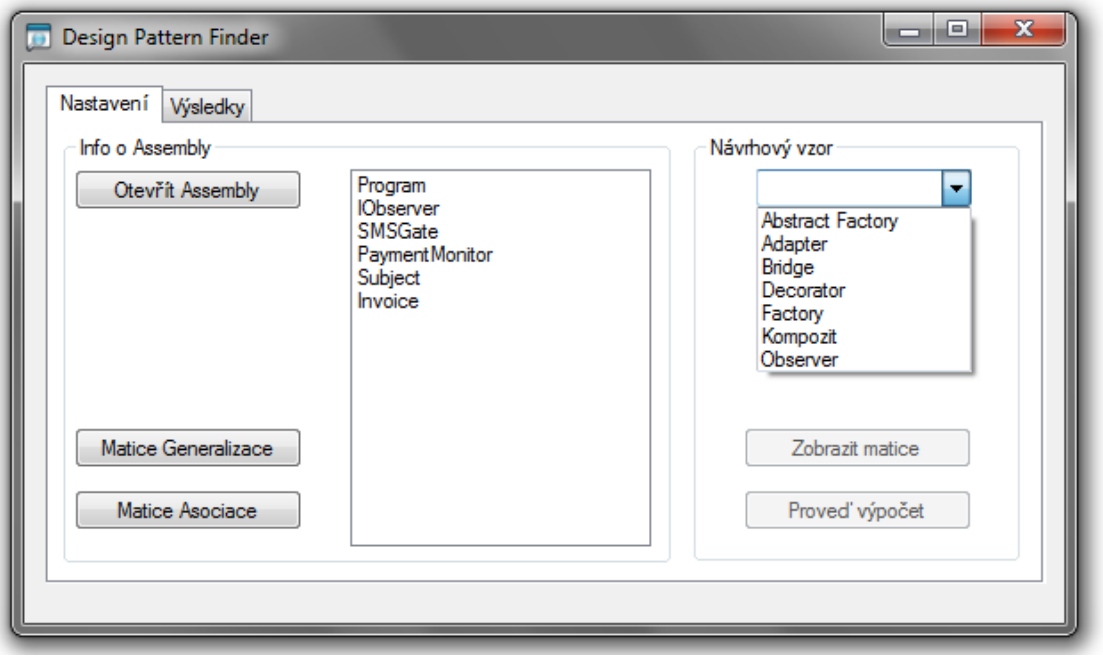


Obrázek 12 : Zobrazení matice generalizace načtené komponenty



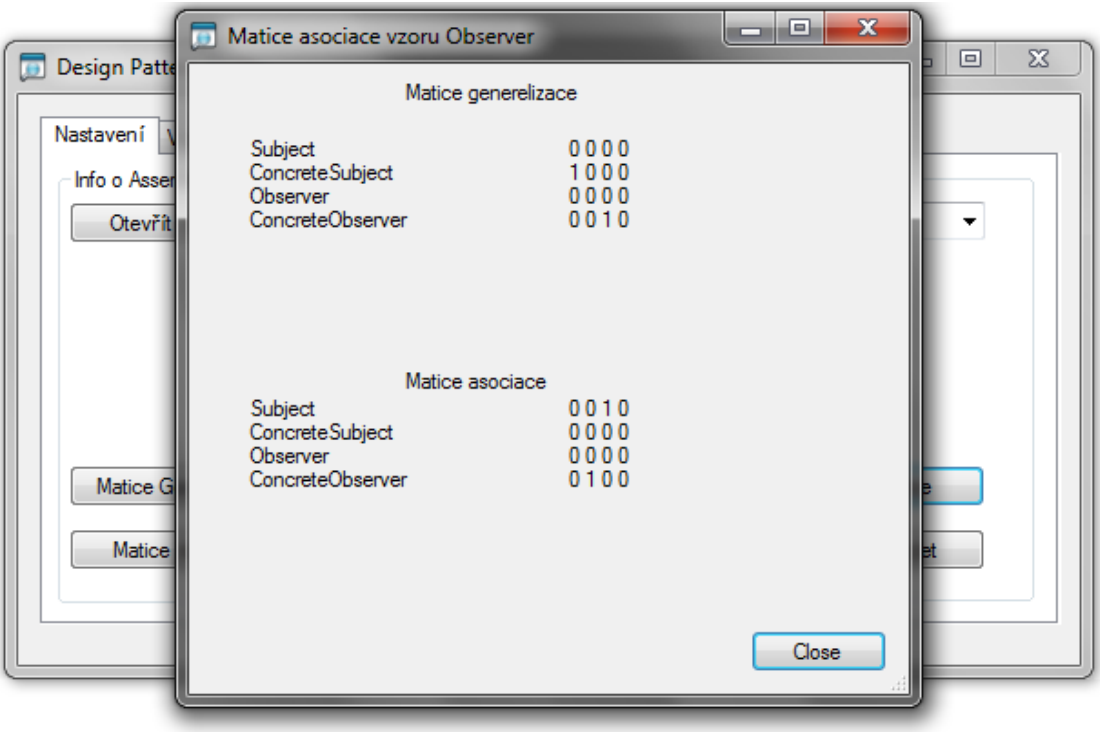
Obrázek 13 : Zobrazení matice asociace načtené komponenty

V další části aplikace můžeme vidět výběr návrhového vzoru, kde si uživatel vybere vzor, který chce v testované komponentě vyhledat. (viz. Obrázek 14)



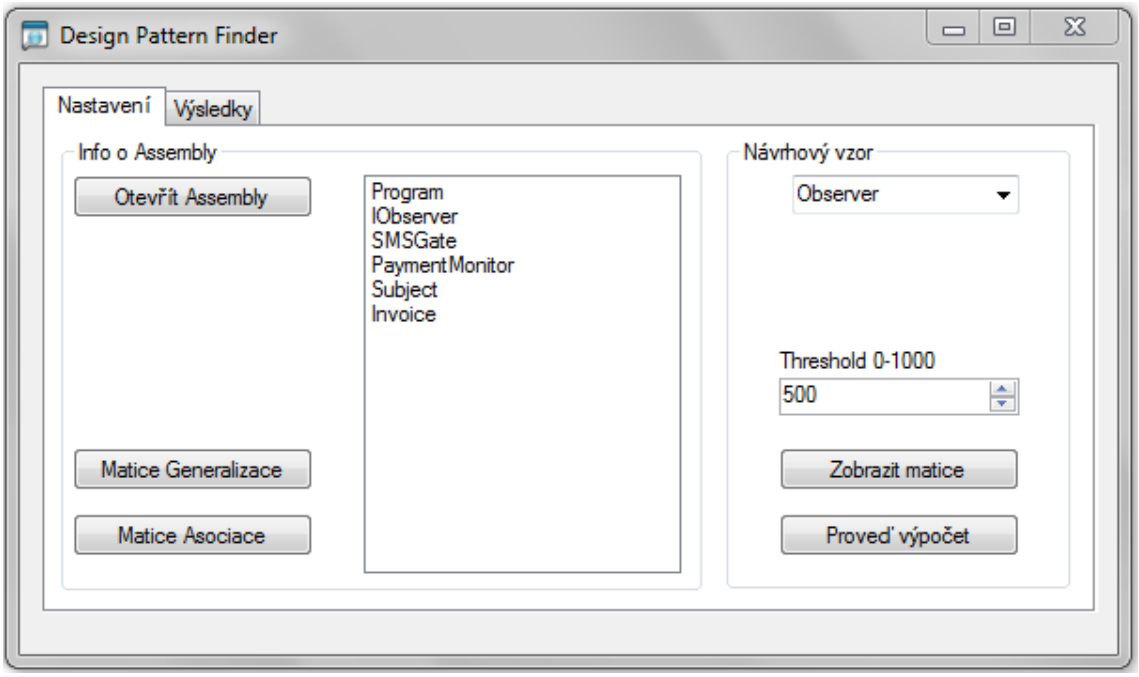
Obrázek 14 : Výběr návrhového vzoru

Po výběru se zpřístupní tlačítko „Zobrazit matice". Díky tomuto tlačítku si může uživatel zobrazit vazby ve vzoru, které jsou popsány stejně jako u matic sousednosti načtené komponenty.



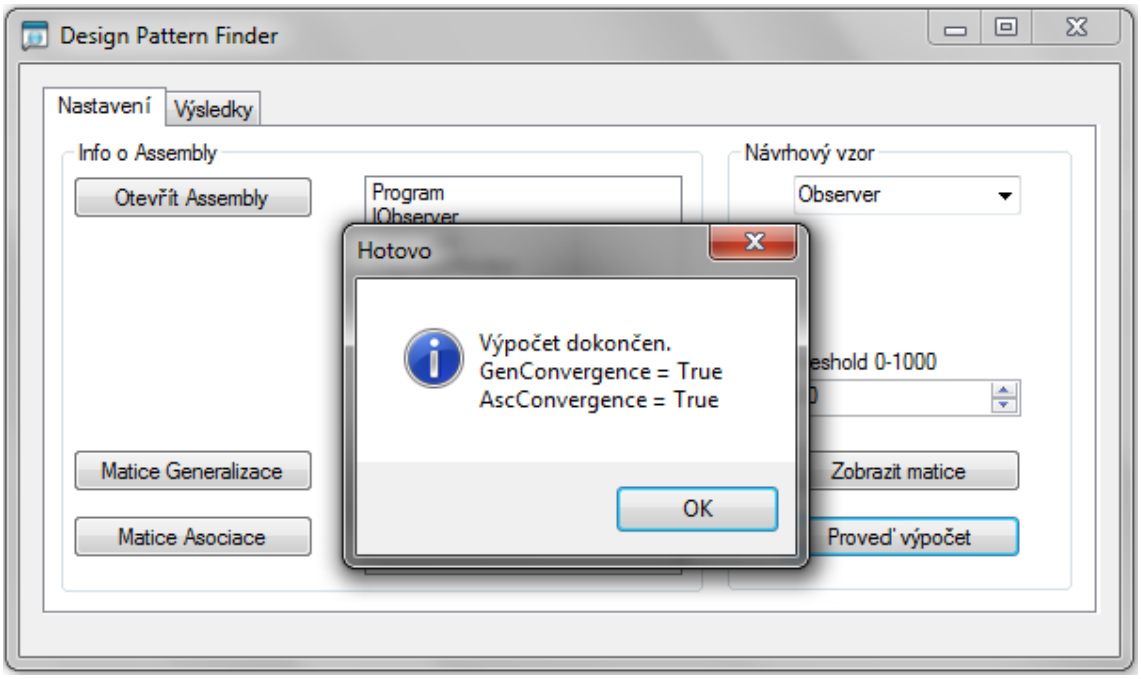
Obrázek 15 : Zobrazení matic sousednosti vybraného návrhového vzoru

Číselník označený jako Threshold zastupuje hodnotu konvergence. Při výpočtu se může stát, že se matice nebude konvergovat k hodnotě 0, ale k jiné. Této číselník tedy používáme k zajištění konvergence. Má-li uživatel načtený systém k analýze a zvolený návrhový vzor, který chce detekovat, zpřístupní se tlačítko „Proveď výpočet“ (viz. Obrázek 16). Po zahájení výpočtu se může zdát, že je aplikace nečinná. Jak dlouho záleží na počtu tříd analyzovaného systému a návrhového vzoru.

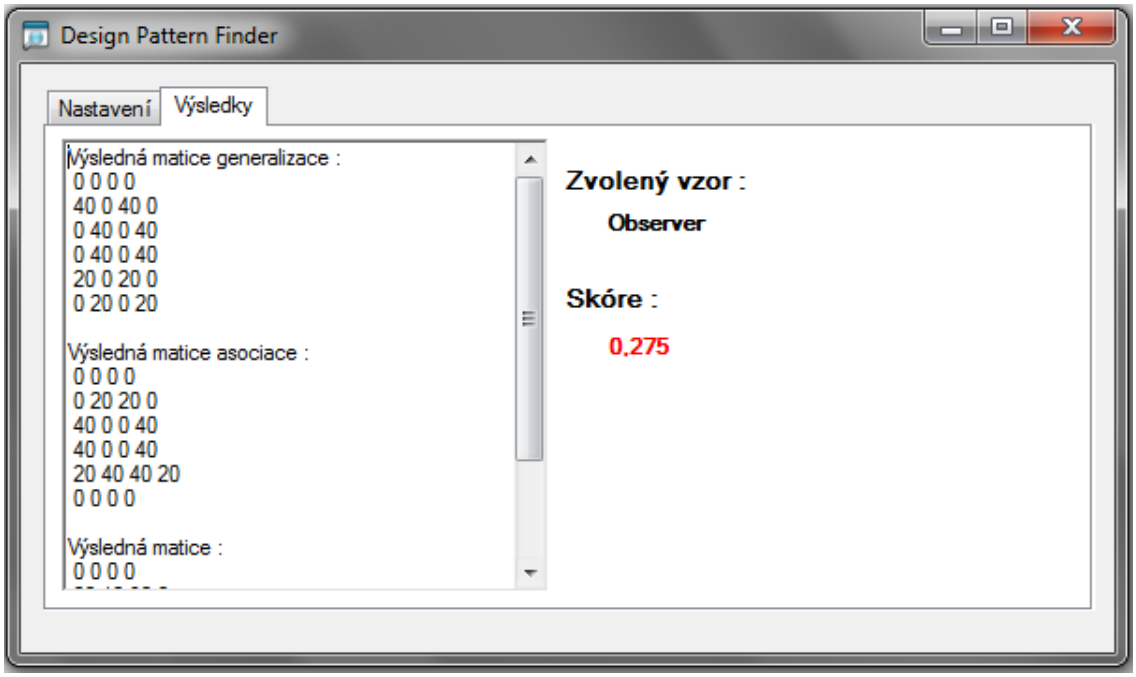


Obrázek 16 : Aplikace s nastaveným systémem k analýze a zvoleným vzorem.

Po dokončení výpočtu je zobrazeno dialogové okno s informací o stavu výpočtu a je zde uvedena iformace o konvergenci matic asociace a generelizace (viz. Obrázek 17) a po něm se aplikace přepne na záložku „Výsledky“.



Obrázek 17 : Konec výpočtu



Obrázek 18 : Karta s výsledky

Po dokončení výpočtu je zobrazena záložka s výsledky (viz. Obrázek 18), kde jsou v levé části zobrazeny výsledné matice podobnosti pro generalizaci a asociaci a výsledná normalizovaná matice. V pravé části je zobrazený zvolený návrhový vzor a celkové skóre, na základě kterého se můžeme rozhodnout, je-li v analyzovaném systému návrhový vzor zastoupen nebo ne. Celkové skóre je počítáno jako podíl součtu maximálních hodnot v matici a počtu tříd daného návrhového vzoru. Maximální součet se počítá tak, že je-li vybrána maximální hodnota tak sloupec i řádek, ve kterém se tato hodnota vyskytuje, se do dalšího výpočtu nezapočítává.

9. Popis implementace

V této části se budu zabývat popisem praktické části bakalářské práce. Na aplikaci Design Pattern Finder, kterou jsem vytvořil, se podíváme ze softwarově inženýrského pohledu. Budu se zabývat analýzou požadavků a pak se podíváme na samostatný návrh aplikace a její klíčové funkce. V programátorské části popíši jednotlivé metody tříd.

9.1. Analýza požadavků

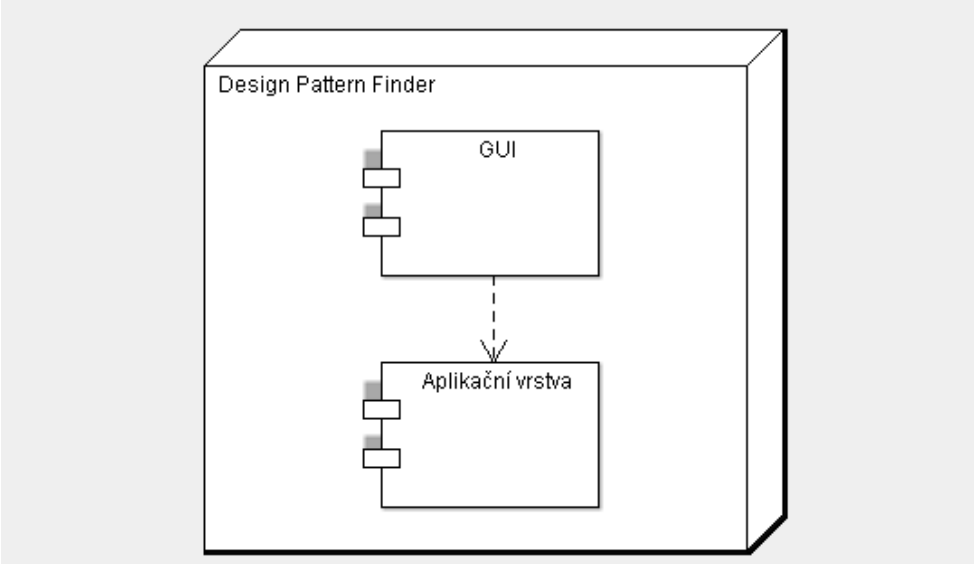
Úkolem aplikace je načíst předkompilovaný program v prostředí .NET Framework a dále pak pomocí reflexe z něj načíst všechny třídy a rozhraní a zjistit vazby typu generalizace a asociace mezi nimi. Na základě zjištěných dat pak sestavit příslušné matice sousednost. Vybrat si jeden z výše uvedených způsobů k porovnání grafů a ten implementovat pro použití detekce návrhových vzorů.

9.2. Návrh aplikace

Návrhem aplikace se myslí diagram tříd a diagram vrstev. Jelikož je samostatná aplikace jednoduchá, je rozdělena pouze do dvou vrstev. Vrstvu obsahující uživatelské rozhraní a vrstvu, která obstarává samostatnou logiku aplikace se všemi vstupy a výstupy.

9.2.1. Diagram vrstev

Jak již bylo řečeno, aplikace je rozdělena do dvou vrstev, které oddělují aplikační logiku a uživatelské rozhraní.



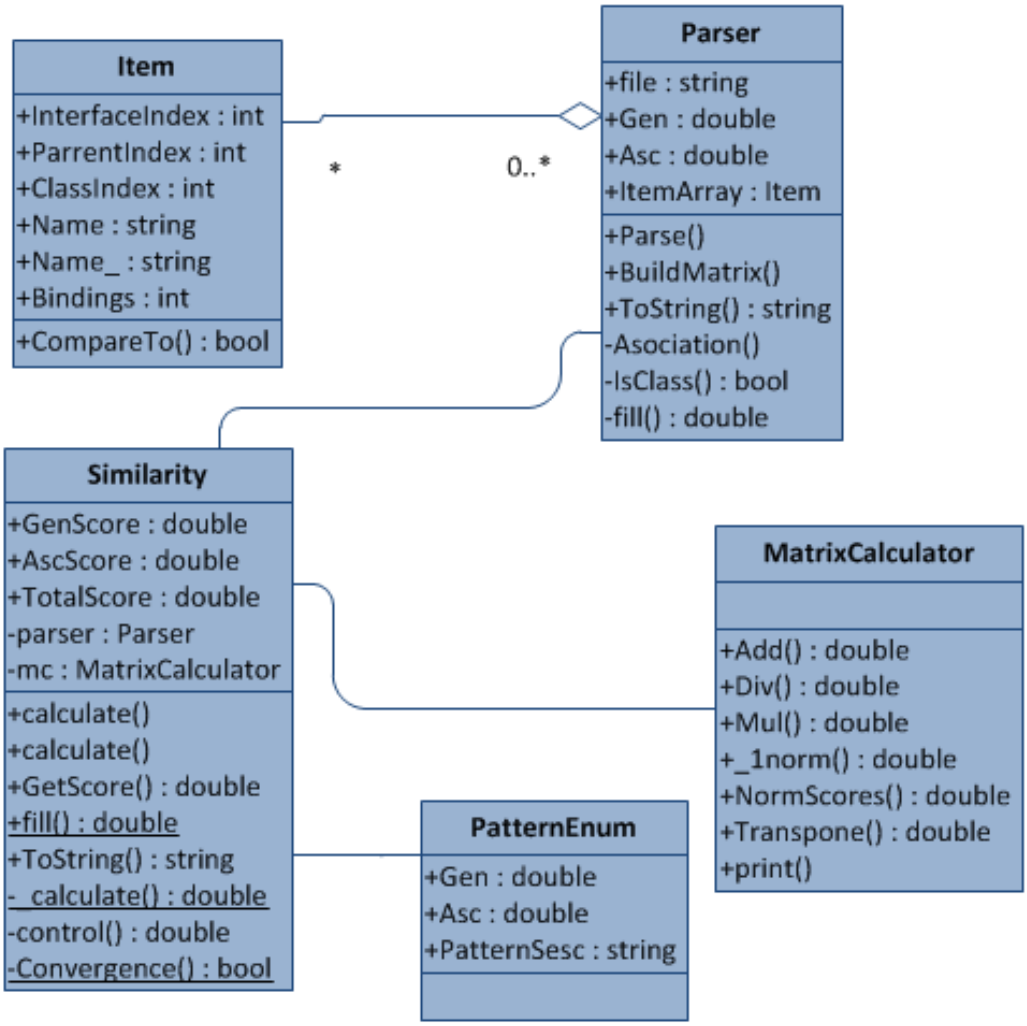
Obrázek 19 : Diagram vrstev

9.2.2. Diagram tříd

V této podkapitole si vyjmenujeme a popíšeme třídy, které aplikace využívá k přečtení informací z načteného kódu, k sestavování matic sousednosti, k matematickým operacím mezi maticemi a k samostatnému výpočtu pomocí Similarity scoring.

Seznam tříd:

- **Parser** – třída se stará o načtení assembly ze souboru, přečtení informací o třídách a datových typech, sestavuje matice sousednosti
- **Item –** pomocná třída, které reprezentuje jednu třídu z načtené assembly a drží o ní informace
- **MatrixCalculator –** matematické výpočty mezi maticemi
- **Similarity –** Samotná implementace algoritmu Similarity Scoring
- **PatternEnum –** Seznam tříd reprezentující jednotlivé vzory



Obrázek 20 : Diagram tříd

9.3. Klíčové funkce

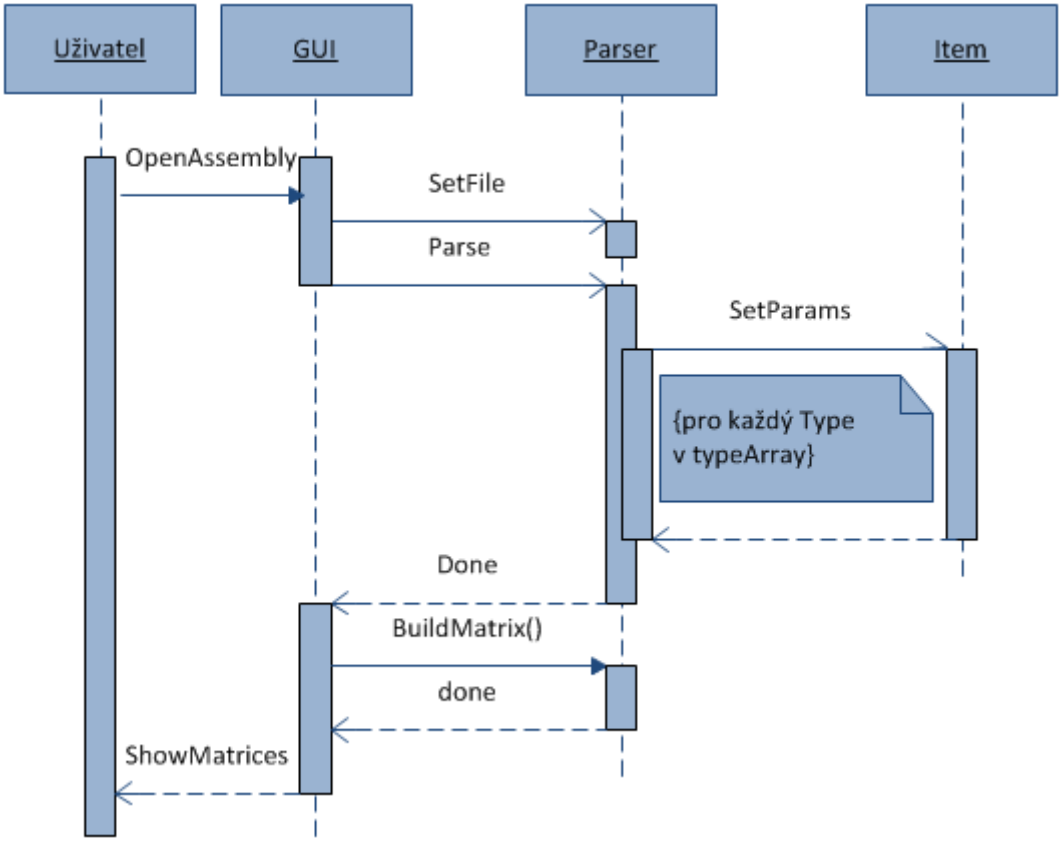
V aplikaci Design Pattern Finder, můžeme najít několik důležitých funkcí. Mezi ty klíčové můžeme zařadit:

- Sestavení matic podobnosti
- Detekce vzoru za použití algoritmu Similarity Scoring

Ke každé z klíčových funkcí uvedu krátký popis a jejich sekvenční diagram, který znázorní časové uspořádání událostí mezi objekty.

9.3.1. Sestavení matic podobnosti

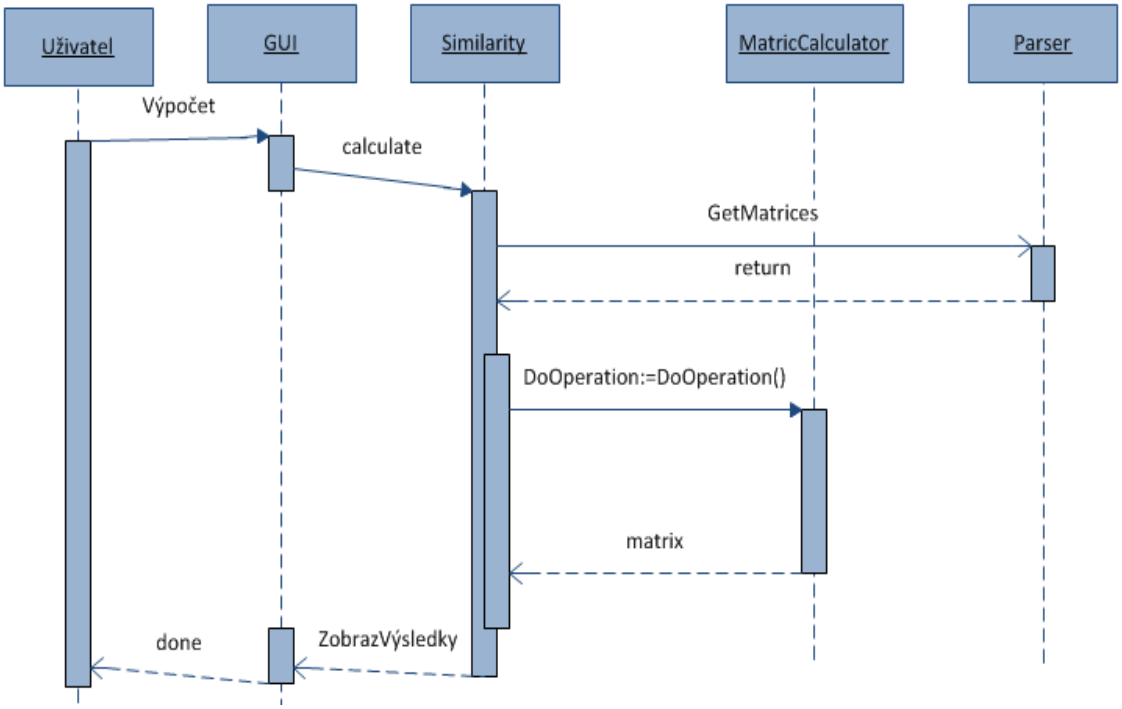
Tento sekvenční diagram popisuje funkci sestavení matic podobnosti. Po zadání cesty k souboru testovaného systému uživatel spustí funkci *Parse()*, která začne prohledávat analyzovaný soubor a zjišťovat třídy v něm obsažené. Ke každé třídě vytvoří pomocný objekt *Item*, který drží informace o třídách, jejich vazbách typu generalizace a asociace. Po dokončení této operace, je spuštěna operace *BuildMatrix(),* která sestaví požadované matice.



Obrázek 21 : sekvenční diagram vytvoření matic podobnosti

9.3.2. Výpočet pomocí Similarity Scoring

Tento sekvenční diagram popisuje funkci výpočtu pomocí similarity scoring. Po načtení testovaného systému a sestavení matic podobnosti může uživatel spustit výpočet. Co se v té chvíli děje popisuje diagram. Po akci uživatele *GUI* zavolá funkci *Calculate()* na objekt *Similarity*. Tento objekt provádí samotný výpočet. Z objektu *Parser* obdriží matice sousednosti a začne počítat podle výše uvedeného vzorce pomocí třídy *MatrixCalculator*, která provádí výpočty mezi maticemi. Po skončení výpočtu *GUI* zobrazí výsledky.



Obrázek 22 : Sekvenční diagram výpočtu Similarity Scoring

9.4. Popis metod

V této podkapitole uvedu popis nejdůležitějších metod, které se v aplikaci vyskytují. Jejich rozdělení je podle tříd, do kterých patří.

9.4.1. ObjectBuilder.Parser

Tato třída má za úkol přečíst všechny třídy, obsažené v analyzovaném systému, zjistit vazby mezi nimi a sestavit matice sousednosti. Její metody jsou:

- *Public void parse()* – metoda vyhledává v analyzovaném systému třídy, vytváří pomocné objekty typu *Item* a vytváří z nich kolekci
- *Private void Association(Type item, List<Item> _it)* – metoda vyhledává vazby typu asociace. Jako vstupní parametr je *Type,* ve kterém hledáme asociaci a

List<Item> - generická kolekce obsahující již vytvořené pomocné objekty. Z této kolekce si metoda vytáhne objekt reprezentující třídu kterou analyzujeme a objekt reprezentující třídu na kterou je nalezená vazba. Tuto vazbu uloží do objektu analyzované třídy.

- *Public void buildMatrix()* – metoda sestaví matici asociace a generalizace
- *Private bool isClass()* – zjistí jestli nalezená vazba je třídou z analyzovaného systému

9.4.2. ObjectBuilder.Item

Instance této třídy se vytváří jako pomocné objekty, které mají za úkol uchovávat informace o nalezených třídách v analyzovaném systému. Nemá žádnou jinou funkci. Vlastnosti třídy jsou:

- *int ClassIndex* – reprezentuje pořadí ve kterém byla třída nalezena a pozici v řádku a sloupci matice sousednosti
- *int ParentIndex* – reprezentuje pozici rodičovské třídy
- *int InterfaceIndex* – reprezentuje pozici rozhraní, které třída implementuje
- *string Name* – vlasní název třídy
- *string Name_* - název oboru názvů (namespace) třídy
- *int[] Association* – pole indexů tříd na které má daná třída vazbu typu asociace

9.4.3. MatrixCalculator.Calculator

Třída provádí matematické operace mezi maticemi. Metody třídy jsou:

- *public double[,] add (double[,] A, double[,]B)* – sečte dvě matice
- *public double[,] mul (double[,] A, double[,]B)* – vynásobí dvě matice
- *public double[,] div (double[,] A, double val)* – vydělí prvky matice hodnotou "val"
- *public double[,] transpone (double[,] A)* – transponuje matici
- *public double _1norm (double[,] A)* – vypočítá řádkovou normu matice
- *public double[,] fill (double[,] A, double[,]B)* – vyplní matici zadanou hodnotou

9.4.4. SimilarityScoring.Similarity

Třída provádí samotný výpočet metodou similarity scoring.

- *Public void calculate(Parser parser, double[,] patternGen, double[,] patternAsc)* – metoda volá výpočet pro jednotlivé matice
- *Public double[,] _calculate(double[,] Code, double[,] Pattern)* – samotná implementace algoritmu similarity scoring
- *Public bool Convergence(double[,] A, double[,]B)* – metoda pro výpočet konvergence matice

10. Experimenty

Součástí této práce jsou experimenty provedené aplikací Design Pattern Finder (DPF). Experimenty se myslí pokusy na vstupních datech ať už speciálně vytvořených k tomuto účelu, tak i na datech reálných. Máme k dispozici data (komponenty) a máme možnost, pomocí reflexe, sestavit vztahy typu asociace a generalizace mezi třídami v komponentě.

V této části si tedy popíšeme pokusy, které byly v rámci aplikace provedeny. Pokusy bychom měli rozdělit do dvou kategorií:

- **Pozitivní**
  - **Pozitivní pravdivý -** TP (true positive)
  - **Pozitivní nepravdivý** – FP (false positive**)**
- **Negativní**
  - **Negativní pravdivý –** TN (true negative)
  - **Negativní nepravdivý** – FN(false negative)

10.1. Pozitivní

Těmito experimenty se myslí vstupní data, u kterých byl návrhový vzor detekován. Tyto výsledky dělíme na pravdivé a nepravdivé:

U pravdivých víme, že návrhový vzor je obsažen ve zdrojovém kódu a metoda detekce prokáže jeho existenci

U Nepravdivých víme, že návrhový vzor není obsažen ve zdrojovém kódu, ale metoda detekce jej přesto v kódu najde.

10.2. Negativní

Těmito experimenty myslíme ty vstupní data, u kterých návrhový vzor detekován nebyl. Tyto výsledky také dělíme na pravdivé a nepravdivé:

U pravdivých víme, že návrhový vzor není ve zdrojovém kódu obsažen a metoda detekce to i potvrdí.

U nepravdivých víme, že návrhový vzor je v kódu obsažen, ale metoda detekce jej neobjeví.

10.3. Vlastní experimenty

Vlastní experimenty byly provedeny na 4 komponentách a na vzorech, které aplikace poskytuje. Jednotlivé experimenty a výsledky jsou popsány v následujících kapitolách. Jako výsledek je uváděn součet maximálních podobnostních koeficientů jednotlivých tříd testované komponenty na třídy zvoleného vzoru a jejich průměr.

V experimentech budou návrhové vzory značeny následovně:

Abstract Factory – AF
Adapter - A
Bridge – B
Composite – C
Decorator – D
Factory – F
Observer - O

10.3.1. Experiment č. 1

První experiment jsem provedl na komponentě speciálně vytvořené pro účel testování aplikace. Jedná se o komponentu reprezentující samotný návrhový vzor. Následující tabulka zobrazuje výsledky detekce, pro každý z nabízených návrhových vzorů

	AF	A	B	C	D	F	O
Průměr	0,148	0,375	0,375	0,583	0,375	0,5	0,437
Součet	0,888	1,125	1,125	1,749	1,125	1,5	1,311

Shrnutí a závěr

O první testované komponentě jsme věděli, že obsahuje návrhový vzor Kompozit, protože byla vytvořena pro demonstraci úspěšnosti zvolené metody. Z výsledků je patrné, že metoda similarity scoring návrhový vzor úspěšně detekovala. Návrhová vzor, Kompozit - C, dostal nevětší bodování. Můžeme tedy tento experiment označit jako pozitivní pravdivý (TP), protože komponenta vzor obsahovala a metoda jej detekovala.

10.3.2. Experiment č. 2

Další testovanou komponentou je komponenta vytvořená studenty. Nevíme jistě, který nebo jestli vůbec nějaký návrhový vzor je v kódu obsažen. Následující tabulka popisuje jednotlivé výsledky:

	AF	A	B	C	D	F	O
Průměr	0,09	0,25	0,175	0,3	0,24	0,437	0,437
Součet	0,54	1	0,7	0,9	0,96	1,311	1,311

Shrnutí a závěr

Z výsledků můžeme zjistit jistou podobnost komponenty s návrhovým vzorem Observer a Factory. Nemůžeme tedy s jistotou říci který návrhový vzor je v komponentě zastoupen. Po prozkoumání výsledných matic podobnosti se můžeme přiklonit k návrhovému vzoru Observer a to z toho důvodu, že výsledná matice pro Observer ukazuje větší míru podobnosti než matice pro Factory.

$$S_{Observer} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 20 & 10 & 30 & 0 \\ 20 & 20 & 0 & 40 \\ 20 & 20 & 0 & 40 \\ 20 & 20 & 30 & 10 \\ 0 & 10 & 0 & 10 \end{pmatrix}$$

$$S_{Factory} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 30 & 0 & 20 & 0 \\ 0 & 20 & 0 & 40 \\ 0 & 20 & 0 & 40 \\ 30 & 0 & 10 & 10 \\ 0 & 10 & 0 & 10 \end{pmatrix}$$

10.3.3. Experiment č. 3

Další testovanou komponentou je komponenta neobsahující žádný návrhový vzor. Tento experiment můžeme zařadit do kategorie Pozitivní nepravdivé, protože jak uvidíme ve výsledkové tabulce, detekční metoda zobrazuje jistou podobnost na návrhové vzory.

	AF	A	B	C	D	F	O
Průměr	0,2875	0,2875	0,275	0,333	0,2375	0,312	0,312
Součet	1,7	1,15	1,1	0,999	0,95	1,248	1,248

Shrnutí a závěr

Experiment ukazuje nepřesnost detekční metody. Výsledky zobrazují jistou podobnost na vzory Composite, Observer a Adapter, což není pravda. V komponentě není obsažen žádný návrhový vzor, byly zde použity nějaké vazby typu asociace/generalizace. Proto tento experiment spadá do kategorie FP.

10.3.4. Experiment č. 4

Poslední experiment vyl proveden také na komponentě, která neobsahuje žádný návrhový vzor. Byla opět použita komponenta vytvořená studenty. V komponentě není použit žádný návrhový vzor a detekční metoda by tuto informaci neměla potvrdit.

	AF	A	B	C	D	F	O
Průměr	0	0	0	0	0	0	0
Součet	0	0	0	0	0	0	0

Shrnutí a závěr

Detekční metoda dokázala, že v testované komponentě není obsažen žádný návrhový vzor.

11. Závěr

V této bakalářské práci jsem se pokusil popsat metody a přístupy k vyhledávání návrhových vzorů v rámci .NET frameworku.

Pro implementace aplikace Design Pattern Finder, která je cílem této práce, jsem vybral a popsal algoritmus Similarity Scoring, který počítá podobnostní skóre dvou matic. Tento algoritmus spadá do kategorie přibližných metod detekce, což experimenty provedené nad vytvořenou aplikací také dokázaly. Ale i přes to, metoda podává relevantní výsledky. Pomocí experimentů jsem zjistil, že zastoupení návrhových vzorů ve studentských projektech je minimální až nulové.

Výsledky této práce by měly sloužit k dalšímu zkoumání v oblasti detekce návrhových vzorů.

Seznam obrázků

Reference

1. Wikipedia. *Wikipedia.org.* [Online] http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor.

2. **Pavlíček, Luboš.** Objektová analýza, návrh a programování. [Online] 16. 6 2005. http://objekty.vse.cz/Objekty/Vzory.

3. **Kučerová, Helena.** [Online] 31. 3 2007. http://web.sks.cz/users/ku/pri/tridy.htm.

4. *Inexact Graph Matching Using Estimation of Distribution Algorithms.* **Bengoetxes, Endika.** 2002, str. http://www.sc.ehu.es/acwbecae/ikerkuntza/these/.

5. **Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, Spyros T. Halkidis.** *Design pattern detection using similarity scoring,.* místo neznámé : IEEE Transactions on Software Engineering,, 2006.

6. Wikipedia. *Wikipedia.org.* [Online] http://cs.wikipedia.org/wiki/Common_Intermediate_Language.